VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV INFORMAČNÍCH SYSTÉMŮ
DEPARTMENT OF INFORMATION SYSTEMS

APLIKACE PRO TÝMOVOU SPOLUPRÁCI

APPLICATION FOR TEAM COLLABORATION

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — **VLADIMIR CHERNENKO**
AUTHOR

VEDOUCÍ PRÁCE — **Ing. BURGET RADEK, Ph.D.**
SUPERVISOR

BRNO 2017

Abstrakt

Tato bakalářská práce se zabývá řešením problému zjednodušeni týmové spolupráce, který se týká komunikace mezí členy týmu a plánování projektů, jeho úloh a možných událostí s projekty spojenými. Na základě provedeného výzkumu a osobních zkušenosti byla navrhnuta webová aplikace, jejímiž hlavními přednostmi jsou jednoduchost ovládání, integrace již existujících užitečných a často využívaných aplikaci a usnadnění komunikace týmu. První část technické zprávy se zabývá popisem návrhu teto aplikace, druhá obsahuje detaily realizace a testování.

Abstract

This thesis deals with solving the problems of simplification of team collaboration, which relates to the communication among the team members and project planning, tasks and possible events, which are related to the projects. Based on a research and personal experience, a web application was designed, whose main advantages was simplicity in use, integration of already existing and frequently used applications and making the team communication easier. The first part of the thesis describes a design of the web application, the second part includes implementation and testing details.

Klíčová slova

Týmová spolupráce, webová aplikace, komunikace v reálném čase, návrh a vývoj aplikace, Django, Websocket, WebRTC.

Keywords

Team collaboration, web application, real time communication, application development and design, Django, Websocket, WebRTC.

Citace

CHERNENKO, Vladimir. *Aplikace pro týmovou spolupráci*. Brno, 2017. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Burget Radek, Ph.D.

Aplikace pro týmovou spolupráci

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením Ing. Radeka Burgeta, Ph.D.. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Vladimir Chernenko
13. května 2017

Poděkování

Rád bych poděkoval Ing. Radeku Burgetovi, Ph.D. za cenné rady, věcné připomínky a vstřícnost při konzultacích a vypracování bakalářské práce.

Obsah

Úvod

V současné době je v rámci vývoje rozsáhlých projektů správné nastavení kolaborace a kooperace týmu zárukou vysoké kvality finálního produktu. Z toho důvodu byly nástroje pro zjednodušení komunikace a organizaci spolupráce týmu vždy velmi důležitou součástí vývoje jakéhokoliv projektu. Nezávisle na tom, nad jakou částí projektu tým pracuje, například vývoj webové aplikace nebo návrh marketingové strategie, organizační problémy vždy zůstávali velice podobné. Plánování jednotných etap budování projektu, s ním souvisejících úkolů, schůzek, hledání řešení a velké množství dalších problémů přináší další ekonomické a časové náročnosti a obtížnosti nejenom pro tým. Nejjednodušším řešením mohla být tabule s různobarevnýma nálepkama, na kterých by byli rozepsány samostatné úkoly a kalendář schůzek. Ale, vzhledem k tomu, že plánování vývoje rozsáhlých projektů je komplexní a náročný proces, jsou v dnešní době nejčastěji využívány webové aplikace. Možnosti, které mohou poskytovat tyto aplikace mohou být různé, od jednoduchého sledování stavu úkolů, až po komunikaci s členy týmu pracujícími na dálku v reálném čase.

Hlavním cílem bakalářské práce je vytvoření aplikace pro týmovou spolupráci. Záměrem práce je definování množiny problémů souvisejících s kolaborací v týmu a následné provedení analýzy existujících aplikací řešících tyto problémy. Další části práce tvoří zhodnocení získaných údajů. Poslední etapou je návrh vlastního řešení a jeho následná realizace.

Bakalářská práce je rozdělena na teoretickou a praktickou část. Teoretická část představuje specifikaci množiny problémů týmové spolupráce, analýzu již existujících řešení a návrh vlastní aplikace. Množství existujících řešení využitých při analýze je založena především na několika populárních, v současné době, neplacených aplikací. Definice množiny problémů je založena na skutečných požadavcích uživatelů, který potřebují pracovat v týmu. Na základě specifikace těchto problémů byl vytvořen návrh aplikace. Koncept vlastního řešení je složen ze dvou základních částí, návrhu uživatelského rozhraní a architektury databázového serveru, který zpracovává požadavky uživatelů. Praktická část představuje realizaci aplikace podle návrhu. Implementace má podobný styl jako návrh, je provedena zvlášť pro uživatelské rozhraní a zvlášť pro databázový server. Součástí pozadí aplikace je návrh a realizace prostředků pro komunikaci v reálném čase. Každá část aplikace pak následně prochází testováním. Finální etapou je zpřístupnění výsledné aplikace na veřejném webu.

V průběhu realizace této aplikace je nutné brát v úvahu již existující standardy. Řešení by mělo splňovat základní požadavky problematiky a navíc nabídnout další možnosti rozšíření. Tato rozšíření musí být navržena pro co nejefektivnější práci uživatele. Hlavní důraz finální aplikace je kladen na jednoduchost uživatelského rozhraní, množství nabízených funkcí a vytvoření soukromého pracovního prostředí pro vývojářský tým.

Kapitola 1

Aplikace pro týmovou spolupráci

Aktuální kapitola definuje základní množinu problémů týmové kolaborace a popisuje provedenou analýzu existujících řešení. Na základě výsledků analýzy bylo provedeno hodnocení existujících aplikaci pro týmovou spolupráci.

1.1 Popis problému

Během vývoje projektu je možné problémy spolupráce v týmu klasifikovat do dvou primárních skupin. První skupina zahrňuje všechny organizační problémy kooperace týmu a druhá souvisí s veškerou komunikací mezi členy týmu.

Životní cyklus projektu je složen z jednotlivých, předem definovaných, etap [5]. Plán každé etapy je představen skupinou cílů a množinou činnosti. Jednotlivé činnosti jsou převedeny na úkoly, které jsou následně rozděleny mezi členy týmu. Každému úkolu muže být přiřazena priorita, časové rozmezí, odpovědná osoba týmu a množina dalších atributů. Do plánu etapy mohou být zahrnovány pravidelné schůzky, během kterých vývojářský tým probírá otázky a problémy tykající se vývoje a jeho organizace. Všechny tyto části jednotlivých etap a etapy samotné musí být dostupné a to tak, aby je mohl pochopit každý člen týmu a mít k nim vždy přístup. Je nutné brát v potaz, že může nastat situace, kdy někteří členi týmu budou pracovat vzdáleně, což přináší další problémy pro zobrazení stavu projektu.

V současné době je komunikace nezbytnou součásti správného fungování týmu. Jednotlivý členové se mohou nacházet v různých městech anebo státech. I když bude mít celý tým možnost sledovat stav projektu a splňovat úkoly podle rozvrhu, potřebuje nástroj pro komunikaci. Příkladem takového nástroje může být obyčejný mail, ale nejčastěji je potřeba komunikovat v reálném čase. Příkladem využití takových nástrojů mohou být online schůzky nebo privátní chaty. Online schůzek se může účastnit i část týmu pracující na dálku, což umožňuje mít aktuální přehled o stavu projektu a sdílet své nápady. Privátní chaty jsou narozdíl od skupinové online komunikace nezbytné pro pohodlné vysvětlení otázek mezi členy týmu.

Řešení všech problémů popsaných výše musí být řešeno na jednom místě, přesněji v jednom systému. Důvodem je zvýšení efektivity jednotlivých členů týmu pomocí organizace soukromého pracovního prostoru. V případě korektní organizace bude mít vývojář vždy přehled o stavu projektu, aktuálních úkolů a případně informaci o tom, s kým může tyto úkoly nebo další otázky probrat. Soukromý pracovní prostor může navíc nabízet volnost z pohledu užití aplikace třetích stran, na které je vývojář navyklý. Příkladem může být Go-

ogle Kalendář. V případě, že vývojářský tým potřebuje mít přistup k společným firemním dokumentům, použití vnějších aplikací nemusí být vyhovující volba, pokud není zajištěn zabezpečený a privátní prostor pro tyto dokumenty. Ve výsledku potřebuje vývojářský tým systém, který řeší následující řadu problému:

- Řízení projektu.
- Řízení úkolů.
- Řízení schůzek.
- Vizuální představa o aktuálním stavu projektu.
- Možnost komunikace v reálném čase. Pro online schůzky můžou to byt video–audio hovory. Privátní chaty jsou příklad textové online komunikace.
- Soukromý pracovní prostor pro jednotlivé členy týmu. Měl by obsahovat možnost integrace aplikace třetích stran.
- Firemní skladiště dokumentů.

1.2 Existující řešení

Jednou z možností, jak efektivně realizovat týmovou spoluprací, tvoří webové technologie. Hlavním důvodem je, že uživatel nepotřebuje instalovat na svůj počítač nebo mobilní telefon žádný dodatečný software. Stačí standardní webový prohlížeč, který je v současné době standardním programovým vybavením téměr každého operačního systému. Další předností využití web technologií spočívá v tom, že uživatelé jíž obvykle mají nějaké zkušenosti s prohlížením internetových stránek, což zrychluje porozumění prostředí aplikace. Jednou z nevýhod pro vývojáře je komplikace s kompatibilitou aplikace napříč prohlížeči. Vývojář musí dávat pozor a být schopný reagovat na případy, kde se funkčnost liší v závislosti na použitém prohlížeči. V dnešní době existuje velké množství projektů, které se snaží vyřešit problém týmové spolupráce v originálním stylu. Dále je představena subjektivní analýza seznamu několika populárních aplikací.

Trello

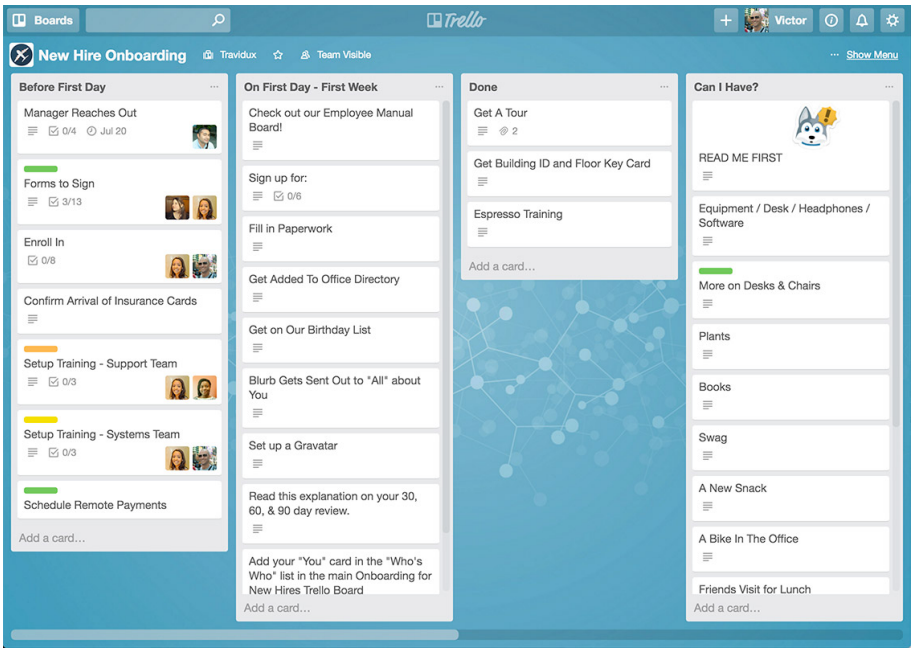
Aplikace Trello[1] byla vyvíjena v roce 2010 společností Fog Creek Software. Základem Trello jsou pracovní zóny, jejichž počet může být nekonečný. Každá z pracovních zón představuje množinu seznamu s úkoly a může být otevřena pro veřejnost, nebo být privátní. Trello navíc obsahuje velké množství integrovaných aplikací třetích stran: appear.in, Box, Dropbox, Evernote, GitHub a etc. Existuje jak verze zdarma, tak i placená verze pro firmy.
Výhody:

- Velké množství integrovaných aplikací třetích stran.
- Široké možnosti konfigurace rozhraní.
- Rozsáhlý funkcionál.

[1] https://trello.com/

Nevýhody:

- Pro některé integrované aplikace je potřeba mít zvlášť účet. Pokud firma preferuje privátní prostředí, je provoz takové aplikace náročnější.



Obrázek 1.1: Příklad pracovní zony v aplikaci Trello [2].

BaseCamp

Basecamp[2] byl vyvíjen v roce 2004 společností Basecamp. Základem je pracovní zóna, která může popisovat jak celý projekt, tak i jeho část. Zóna je rozdělena na 5 různých komponent. První je „Campfire“. Pomocí „Campfire“ je uživatel schopen komunikovat s některým z uživatelů, kteří jsou napojeni na tuto pracovní zónu. Druhá komponenta je „Message board“, do které může uživatel napsat článek nebo zprávu a zbytek je schopný komentovat. Třetí část je „To-dos“, do které je možné vkládat úkoly. Další komponentou je „Schedule“, která popisuje důležité aktualizace a blížení se k finálním terminům. Poslední část tvoří „Docs & Files“.
Výhody:

- Pohodlné rozhraní.
- Dostačující funkcionalita pro jednoduché projekty.

Nevýhody:

- Absence privátních konverzací.
- Absence audio–video konverzací.
- Malé množství integrovaných aplikací třetích stran.

[2] https://basecamp.com/

Obrázek 1.2: Přiklad pracovní zóny v aplikaci Basecamp [1].

Slack

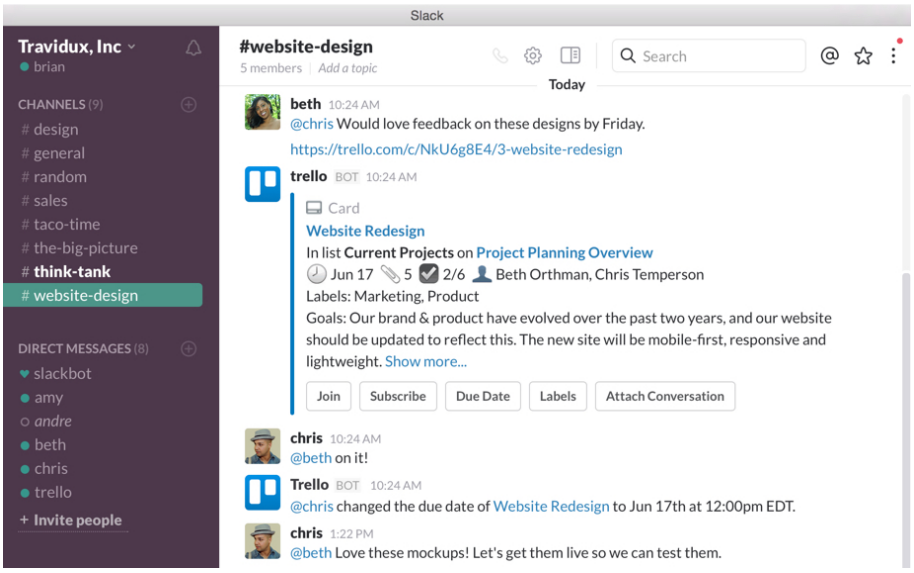
Aplikaci Slack[3] vyvinul v roce 2013 Stewart Butterfield . Slack je složen ze tří částí: „Channels“, „Direct messages“ a „Calls“. „Channels“ jsou kanály pro komunikaci a mouhou být privátní i veřejné. „Direct messages“ je přímá komunikace s uživatelem. „Calls“ je audio–video komunikace.
Výhody:

- Ideální pro komunikaci v týmu.

Nevýhody:

- Neřeší řízení projektů a úkolů.

[3] https://slack.com/



Obrázek 1.3: Příklad komunikačního rozhraní v aplikaci Slack [3].

KanbanFlow

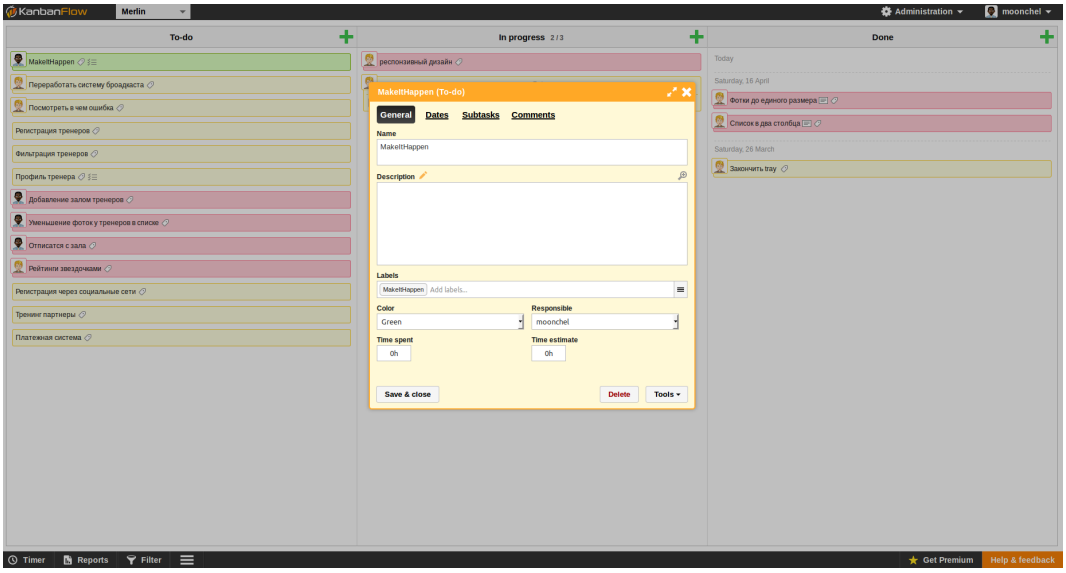
Základem aplikace KanbanFlow[4] je rozdělení každé z pracovních ploch na 3 časti. Každá pracovní plocha může popisovat celý projekt nebo jeho část. První část slouží pro úkoly, které je potřeba udělat, další pro úkoly, které se už řeší a poslední pro dokončené úkoly. Ke každému z těchto úkolů může uživatel zanechat komentář, připojovat podúkoly nebo nastavovat data dokončení a priority.
Výhody:

- Pro malé týmy je jednoduchý na řízení projektů a úkolů.

Nevýhody:

- Neobsahuje souborové skladiště.
- Absence možnosti komunikace.

[4] https://kanbanflow.com/



Obrázek 1.4: Přiklad komunikačního rozhrání v aplikaci KanbanFlow [4].

JIRA

Jira[5] je software, který pomáhá hledat chyby a problémy během tvorby jiných projektů, a nebo ulehčuje řízení týmu. Byl vyvinut společností Atlassian[6]. I když nepatří mez neplacené nástroje, je velmi populární. Ale cena může být nízká, pokud se firma rozhodné koupit JIRA jako standalnone aplikaci.
Výhody:

- Řízení projektu.
- Klientská podpora.
- Sdílení komunikace, informací a dokumentů v týmu.
- Rozdělení úkolů podle priorit.

Nevýhody:

- Neexistuje verze zdarma.

Google aplikace

Google poskytuje celou sadu užitečných aplikací pro řešení řady problémů týmové spolupráce. Populární aplikace jsou Google Calendar a Google Drive. Integrace těchto aplikací výrazně rozšiřuje funkcionalitu jakéhokoliv projektu.

1.2.1 Hodnocení existujících řešení

V současné době existuje poměrně hodně aplikací pro usnadnění týmové spolupráce, ale ne každá z nich může uspokojit všechny požadavky týmu. Na druhou stranu, kombinací

[5] https://www.atlassian.com/software/jira
[6] https://www.atlassian.com/

některých aplikací, například Slack a Basecamp, je možné docílit výrazně vyšší efektivity týmu, vyřešit problémy komunikace a organizace pracovního prostředí. Dále je uvedena tabulka shrnující, jaké problémy týmové kolaborace řeší jednotlivé aplikace.

Název	Audio/Video Conference	Zprávy	Skladiště Dokumentů	Kalendař	Řízení projektů	Řízení úkolů
Trello	Ne	Ne	Ano	Ne	Ne	Ano
Basecamp	Ne	Ano	Ano	Ano	Ano	Ano
Slack	Ano	Ano	Ano	Ne	Ano	Ano
KanbanFlow	Ne	Ne	Ne	Ano	Ano	Ano

Kapitola 2

Architektura výsledné aplikace

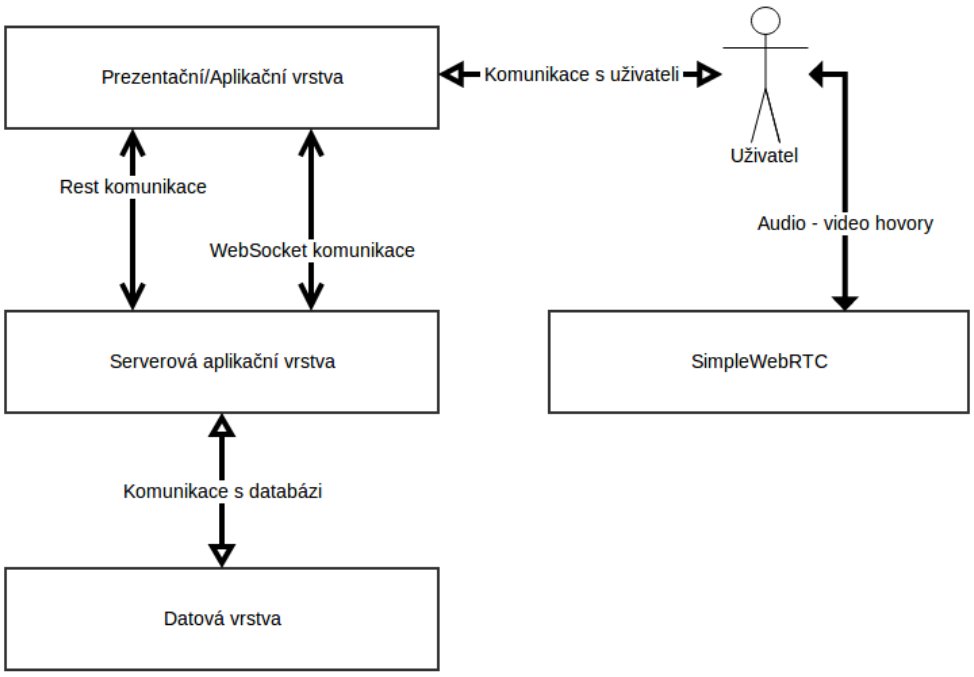
Architektura aplikace je složena z několika vrstev. Vícevrstvá architektura v softwarovém inženýrství označuje aplikace, kterých funkčnost netvoří jednolitý program, ale více navzájem spolupracujících vrstev. Každá z těchto vrstev obvykle běží na různých výpočetnícj infrastrukturách. Pro výslednou aplikaci byla využita třívrstvá architektura. Je znázorněna na obrázku 2.1.

Hlavním úkolem prezentační vrstvy je přijetí a zpracování dat od aplikační vrstvy, a jejich následné zobrazení uživateli. Zobrazení ve výsledné aplikaci je implementováno ve stylu SPA (Single Page Application) či jednostránkové aplikace. Hlavní funkce SPA je synchronizace stavu objektů mezi datovou a prezentační vrstvou. V průběhu prvního příchodu uživatele na stránku prohlížeč stahuje celou aplikaci, a následné jenom stavy jednotlivých objektů.

Cílem aplikační vrstvy je zpracovní požadavků prezentační vrstvy a komunikace s vrstvou datovou, takže se ve výsledné aplikaci skládá ze dvou části. První je REST aplikace, která zpracovává požadavky jednostránkové aplikace a provádí synchronizace stavů objektů přes datovou vrstvu. Komunikace probíhá ve formátu JSON. Registrace, autentizace a autorizace je realizovaná pomoci pohledů, které jsou součástí návrhového vzoru MVC (Model–View–Controller). Pohled v tomto případě přijímá data od SPA a provádí potřebné operace. Druhá část aplikační vrstvy představuje implementaci textové komunikace v reálném čase.

Databázová nebo datová vrstva je zpravidla představena specializovaným databázovým serverem. Na začátku byli definovány všechny modely podle specifikace požadavků na aplikaci. Následně byl vytvořen a realizován databázový návrh. Databázová vrstva zpracovává požadavky aplikační vrstvy a provádí operace ukládání, správu a zabezpečení fyzických dat. Jako databázový nástroj, byl pro aplikaci vybrán SQLite.

Z důvodu zmenšení počtu použitých hardwarových prostředků a samotného rozsahu aplikace, je zprovozněna jenom na jednom serveru.



Obrázek 2.1: Architektura aplikaci.

Kapitola 3

Uživatelské rozhraní

Grafické uživatelské rozhraní nebo Graphical User Interface (GUI) je seskupení technologií a prostředků, které dávají uživateli možnost komunikovat s aplikací a počítačem. V relaci s webovými aplikacemi se především jedná o textové nebo grafické prvky a jejich rozmístění na stránce. Tato skupina ovládacích prvků umožňuje uživateli práci s aplikací. Získává od něho potřebná vstupní data a prezentuje výsledky zpracovaných dat, většinou prostřednictvím webového prohlížeče. [18]

3.1 Návrh

Proces tvorby návrhu uživatelského rozhraní se dá shrnout do několika fází. První fáze je vytvoření strategie, která popisuje, jaké výhody přináší aplikace pro uživatele. Na začátku je nezbytné porozumět, kdo je uživatel, který bude aplikaci využívat. Ve výsledku je vytvořen jeho profil a stanovena cílová skupina. Druhá část této fáze je analýza výhod, které aplikace přináší firmě nebo její tvůrčí. Vzhledem k tomu, že cílem bakalářské práce je vytvoření nekomerční aplikace, nebude analýza výhod provedena. Druhá fáze definuje specifikace požadavků. Hlavním úkolem teto fázi je specifikace možnosti a funkci, která by mělo uživatelské rozhraní pokrývat. Zatím ovšem bez ohledu na to, jak bude funkce vypadat a fungovat. Výsledkem je seznam funkcí s korespondujícím popisem, který je vytvořen na základě požadavků uživatelů. Třetí fázy tvoří návrh struktury. Zabývá se především interakčním designem, neboli interakcí mezi jednotlivými definovanými funkcemi. Důležité v této fázi je porozumět jak budou uživatelé procházet jednotlivými kroky, až po splnění určitého úkolu a jak bude rozdělen obsah aplikace vzhledem k funkčním vazbám. Po skončení této fáze by měli být vytvořené „use case“ diagramy [21] nebo diagramy „Případů užití“, které reprezentují případy užití a nejčastější scénáře, reakce aplikace na operaci uživatelů. Poslední etapa v návrhu uživatelského rozhraní je vytvoření kostry nebo drátěného modelu rozhraní.

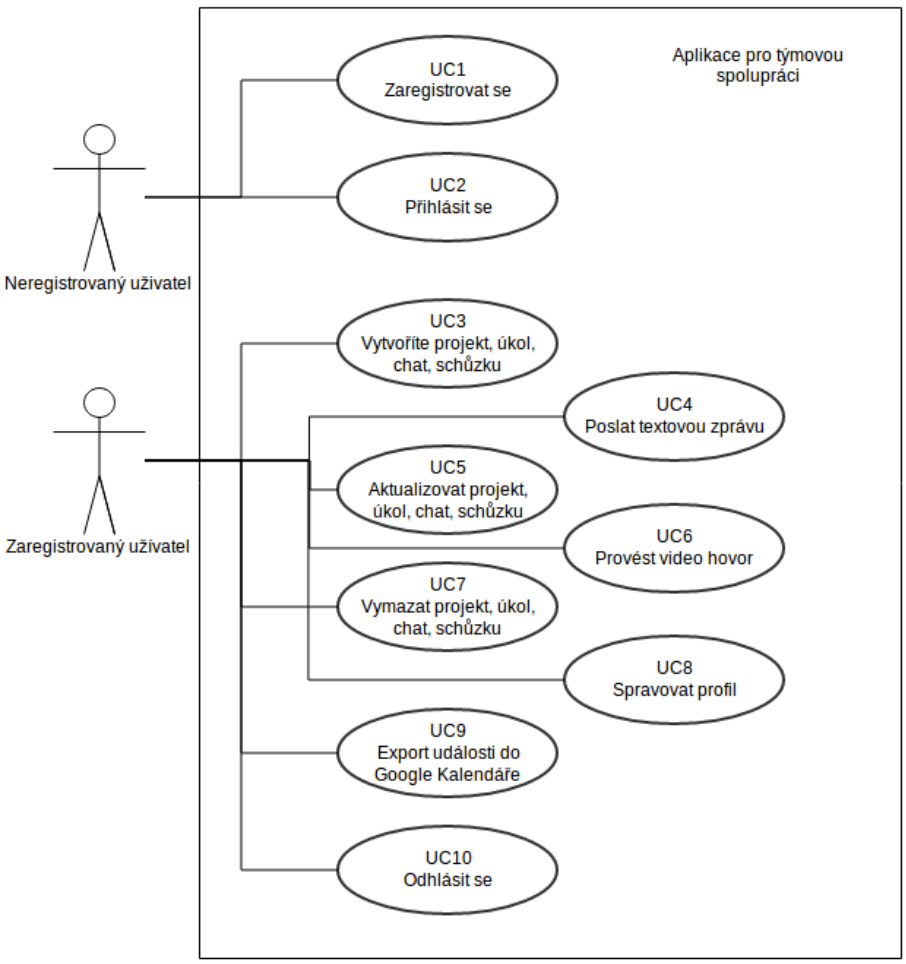
3.1.1 Analýza a specifikace požadavků

Cílová skupina se skládá z lidí, kterým je velice často potřeba pracovat v týmu nad rozsáhlými projekty. Jsou obvykle seznámený s velkým množstvým web aplikací a představuji skupinu zkušených uživatelů. Pro nich je důležitá široká škála funkcionality aplikace a intuitivní rozhraní. Na základě požadavků těchto uživatelů jsou popsaný problémy v sekci 1.1. Po provedení specifikace těchto požadavků, tedy definování podproblémů z každé skupiny, byl vytvořen seznam specifikací:

- Zobrazení seznamu projektů, úkolů, událostí, souborů nebo chatů.
- Formuláře pro vytvoření, mazání, aktualizace projektů, úkolů, událostí, souborů nebo chatů.
- Export událostí.
- Textová komunikace a video–audio hovory.
- Zobrazení nových notifikací.

3.1.2 Návrh struktury

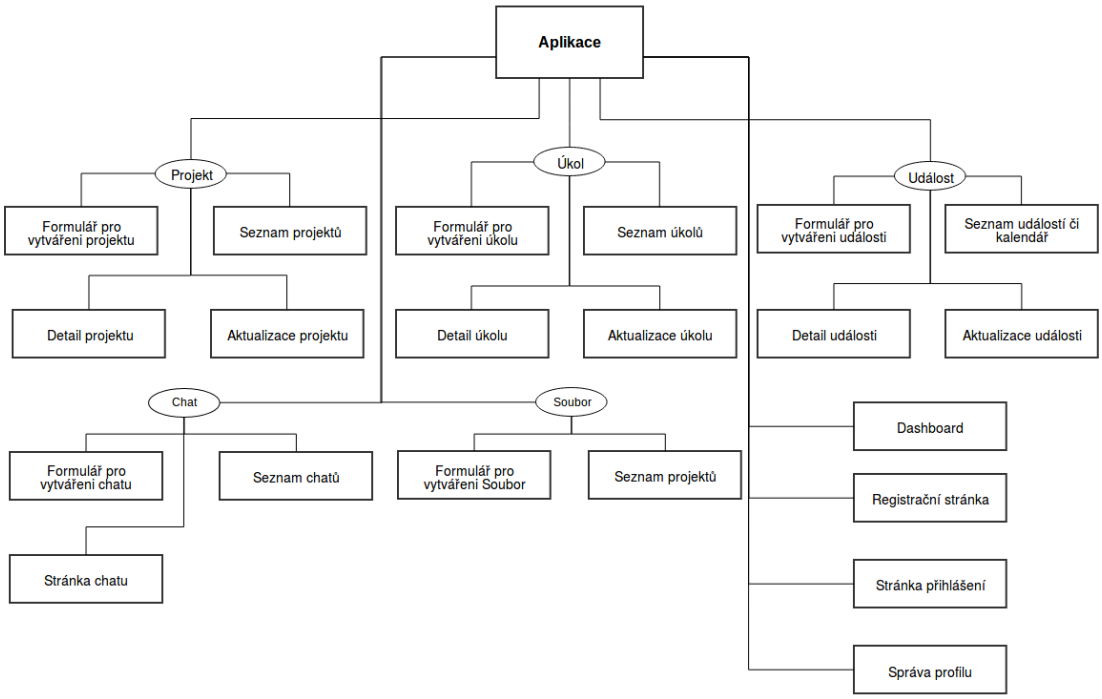
Pro vytváření „use case“ diagramu je nutné pochopit jaké typy uživatelů v aplikaci budou a jaké práva budou přiděleny každému z nich. V aplikaci jsou dva typy uživatelů. Nezaregistrovaný uživatel nemůže provádět žádné operace, s výjimkou registrace nebo přihlášeni. Takový postup byl vybrán s ohledem na to, že všechna data a informace o projektu nad kterými pracuje tým jsou privátní. Po ukončení registrace uživatele jsou dostupné všechny funkce aplikace. Seznam funkcí je možné vidět na obrázku níže.



Obrázek 3.1: Diagram případů užití.

3.1.3 Kostra uživatelského rozhraní

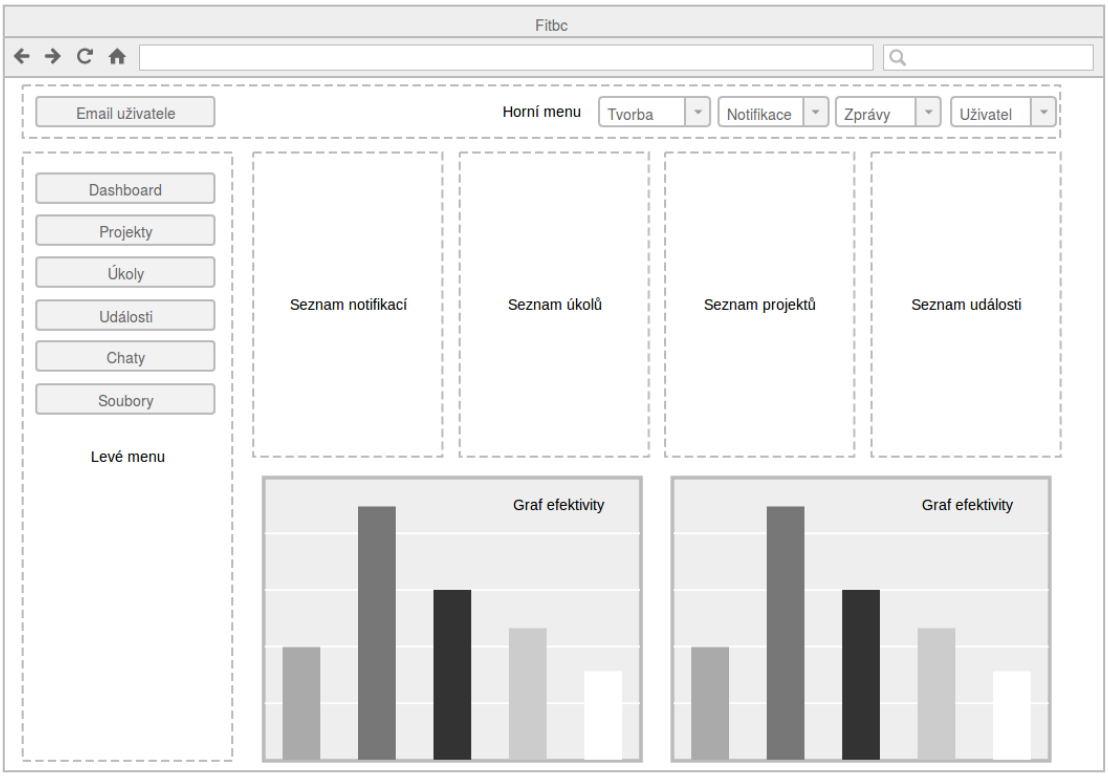
Na základě množiny operací, které může provádět uživatel byla vytvořena kostra uživatelského rozhraní. Některé části aplikace jsou sdružené do skupin, například množina stránek souvisících s úkoly nebo projekty. Kostra je zobrazena na obrázku dole.



Obrázek 3.2: Kostra uživatelského rozhraní.

Jeden z nejdůležitějších komponent je Dashboard. Představuje stránku obsahující všechny aktuální informace o stavech všech projektů, úkolů nebo událostí, ve kterých uživatel působí nebo je účastníkem. Jeho drátěná kostra je zobrazena na obrázku 3.3. Jak je vidět, rozhodl jsem se mít dva menu. Levé menu obsahuje odkazy na Dashboard a seznamy projektů, úkolů, událostí, chatů a souborů. Horní menu se skládá z několika dalších vypadávajících menu:

- Menu pro výběr objektu pro následné vytváření.
- Menu pro zobrazení seznamu notifikací.
- Menu pro zobrazení seznamu zpráv.
- Menu pro přechod do správy profilu a nebo odhlášení.



Obrázek 3.3: Drátěná kostra Dashboard komponenty.

3.2 Implemetace

Realizační etapa začíná po připravení všech drátěných modelů. Představuje tvorbu vizuálního designu nebo vzhledu jednotlivých komponent stránky. Po dokončení realizace tvorby uživatelského rozhraní přichází fáze testování. Testuje se vždy finální verze rozhraní. Testování může byt provedeno pomocí uzavřené cílové skupiny uživatelů a nebo speciálních analytických nástrojů. Příkladem můžou být teplotní mapy nebo nahrávání uživatelské aktivity. V této fázi je důležitá zpětná vazba od uživatelů, na základě které jsou pak promítány změny do fází tvorby kostry nebo do etapy vytváření vzhledu.

3.2.1 Použité technologie

Při vývoje komplexních projektů jsem měl možnost využít existující knihovny. Dále jsou popsýny jenom ty nejdůležitější z nich.

Bootstrap

Bootstrap[1] je sada nástrojů pro tvorbu webu a webových aplikací. Je jednoduchá v použití a přístupná k volnému stažení. Obsahuje šablony založené a vytvářené pomocí HTML, CSS a JavaScript, sloužící pro úpravu formulářů, typografie, navigace, tlačítek a dalších komponent rozhraní.

[1]Bootstrap – http://getbootstrap.com

Vuejs

Vuejs[2] je progresivní framework pro vývoj uživatelských rozhraní. Oproti jiným frameworkům je základní myšlenka Vue přizpůsobovat se situaci bez velkých změn v projektu. Jádro knihovny je zaměřeno jen na viditelné vrstvě a navíc je jednoduché do něj integrovat další knihovny nebo frameworky. Je velmi efektivní při vývoji jednostránkových aplikací.

Vue Router

Vue Router[3] je knihovna pro framework Vue, která umožňuje vyřešit URL adresace.

Vuex

Vuex[4] je knihovna, která obsluhuje globální stav Vue aplikace.

Textový editor Quill

Quill[5] je aplikace v jazyce Javascript, která nabízí široké možnosti formátování textu přímo ve webovém prohlížeči. Je jednoduchý pro použití, přístupný k volnému stažení a multiplatformní.

Websocket

Websocket [20] vychází ze starších technologií jako AJAX (Asynchronous JavaScript and XML) či Comet (dlouhodobé vyhrazené HTTP spojení) a nabízí programátorům jednoduché rozhraní pro navázání spojení a vzájemnou výměnu zpráv mezi klientem a serverem. Na straně klienta jsou WebSockets implementovány v JavaScriptu jako třída WebSocket.

WebRTC

WebRTC [9] (anglicky Web Real–Time Communication) je definice API poskytujícího podporu pro video chat a sdílení souborů mezi jednotlivými uživateli a telefonní hovory. API je poskytováno pro aplikace, které je možné spustit ve webovém prohlížeči. Autorem pracovní verze API je W3C (World Wide Web Consortium).

SimpleWebRTC

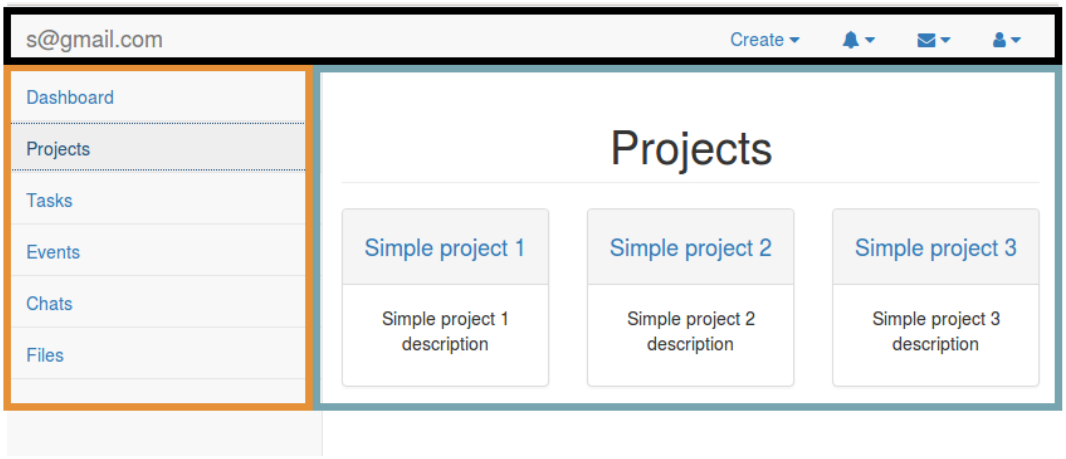
SimpleWebRTC[6] je knihovna, která ulehčuje práci s WebRTC. Obsahuje funkce pro vytvoření video hovoru, sdílení dokumentu a pracovní plochy počítače nebo mobilního telefonu.

3.2.2 Implementace komponent

Pro realizaci návrhu uživatelského rozhraní byla každá stránka rozdělena na menší části či komponenty. Každá z těchto komponent představuje logický celek, izolace kterého usnadňuje realizaci a podporu aplikace. Příkladem může být rozdělení stránky seznamu projektů. Přestože se stránka uživatelovi zobrazuje jako celek, pro vývojáře je jednoduší ji chápat jako

[2] Vuejs – https://vuejs.org/
[3] Vue Router – https://router.vuejs.org/en/
[4] Vuex – https://vuex.vuejs.org/en/
[5] Quill – https://quilljs.com/
[6] SimpleWebRTC – https://simplewebrtc.com/

množinu komponent, viz na obrázku 3.4. Horní menu, které je označeno černou barvou na obrázku, slouží k tomu, aby uživatel mohl přecházet na stránky vytváření objektů, odhlášení nebo aktualizace profilu. Další funkcí horního menu je zobrazení notifikace a nových zpráv. Levé menu, které je označeno oranžovou barvou, slouží k zobrazení odkazů na seznamy objektů. Poslední a nejdůležitější komponenta na stránce zobrazuje seznamy a detaily jednotlivých objektů, formuláře atd. Vzhledem k tomu, že funkce, které realizují komponenty, jsou logicky nezávislé, bylo rozhodnuto vyvíjet každou komponentu samostatně.



Obrázek 3.4: Rozdělení stranky seznamu projektů na komponenty.

Implementace jednotlivých komponent byla provedena s použitím frameworku VueJs. Každá komponenta ve Vue se skládá ze dvou části. První je šablona pro zobrazeni, která je napsána v jazyce HTML. Do šablony může vývojář vkládat cykly, podmínky nebo složité datové struktury. Příkladem komponenty je zobrazení seznamu projektů 3.1. V šabloně je využita podmínka pro kontrolu existence projektů. V případě existence se provede cyklus pro zobrazení jednotlivých objektů v seznamu. V opačném případě uživatel uvidí nadpis „No projects found“. Druhá část komponenty je Javascript objekt, který zpracovává stav komponenty. Pojem „stav“ v kontextu Vue je složen z vnitřních dat komponenty, reakce na vstup uživatele a na data, které přichází ze serveru. Schéma vnitřních dat je popsána funkci „data“. Nahraní seznamu projektů do vnitřních dat komponenty prochází v fáze životního cyklu „mounted“. Popis všech fází se dá najít v dokumentaci [23]. Nejdůležitější pro danou aplikace jsou tři stavy:

- created – stav který popisuje moment inicializace komponenty. V tomto stavu ještě není zobrazena na stránce, a proto je možné v tom stavu stáhnout data a pak se komponenta zobrazí s daty.
- mounted – stav ve kterém se komponenta zobrazuje na stránce. Kvůli tomu, že jakákoliv změna vlastních dat komponenty se projeví hned na zobrazení, je možné stáhnout data i v tomto stavu.
- destroyed – fáze, v průběhu které je komponenta vymazána ze stránky. Tento stav je využít v části s video hovory hlavně kvůli tomu, že pokud uživatel opustí stránku s chatem, tak se musí odhlásit od přijmu jakýchkoliv zpráv a video pozvánek.

```
<template>
<div class="row text-center">
  <h4 v-if="objects.length == 0" class="text-info">
  No projects found</h4>
  <div v-else class="col-xs-4" v-for="project in objects">
    <div class="panel panel-default">
      <div class="panel-heading">
        <router-link :to="{ name : 'project-detail',
                params : { name : project.unique_name }}">
          <h4>{{ project.name }}</h4>
        </router-link>
      </div>
      <div class="panel-body" v-html="project.description"></div>
    </div>
  </div>
</div>
</template>

<script>
export default {
  data () { return { objects : [], meta : null } },
  mounted () {
    this.$http.get('project').then(response => {
      this.meta = response.data.meta
      this.objects = response.data.objects
    })
  }
}
</script>
```

Listing 3.1: Komponenta seznamu projektů.

Pokud vezmeme atribut „objects“, je možné si všimnout, že se jenom zobrazuje a nemůže být změněn uživateli přímo. Takový návrhový vzor se nazývá „one way data binding“, a využívá se jenom pro zobrazení dat. V případě existence atributu komponenty, který je závislý na vstupu uživatele, a zároveň je zobrazen někdy na stránce, se používá „two way data binding“. Navíc každá komponenta může obsahovat metody, reagující na vstup uživatele a získávání dat ze serveru. V projektu byly využity oba dva vzory, první pro zobrazení všech dat, které přichází ze serveru, a druhý pro zpracování všech formulářů.

3.2.3 Globální data

Pro zachování globálních dat, například indikátoru, který popisuje jestli je uživatel přihlášen, je využita knihovna Vuex. Je založena na principu globálního skladiště, které je automaticky integrováno do všech existujících komponent. V projektu jsou v něm uložené různé indikátory, email přihlášeného uživatele a název komponenty. Název se zobrazuje uživatelovi jako nadpis na stránce, pro lepší orientaci v aplikaci. Změny stavu je možné provádět jenom pomocí „mutací“, a pro přistup k datům je vhodné využívat „getters“ atribut.

```
export default new Vuex.Store({
  state : { header : '', isLoggedIn : true,
    user : { email : '', }, },
  mutations : {
    login (state) { state.isLoggedIn = true; },
    logout (state){ state.isLoggedIn = false;
      state.user.email = ''; },
    set_user(state, email) { state.user.email = email; },
    set_header(state, new_header) { state.header = new_header; }
  },
  getters : {
    isLoggedIn : state => { return state.isLoggedIn; },
    user_email : state => { return state.user.email; },
    header : state => { return state.header; }
  }
})
```

Listing 3.2: Skladiště pro uložení stavu aplikace.

3.2.4 Formuláře a URL adresace

Aplikace obsahuje velké množství různých formulářů. Aby nebylo potřeba psát vlastní HTML kód pro každý z nich, byla vytvořena generická komponenta, která příjme na vstup slovník se všemi poli formuláře a funkce, která musí být zavolaná v případě potvrzení formuláře. Komponenta je schopná vytvořit formy se všemi základními typy vstupních polí a navíc obsahuje speciální pole pro Quill editor.

V případě, že je uživatelské rozhraní představeno jednostránkovou aplikaci, musí být vyřešena URL adresace, která v této aplikaci byla realizována pomocí knihovny VueRouter. Základní myšlenkou je hledání komponent odpovídajících URL adrese a jich následné zobrazení. V průběhu implementace aplikace byli navíc využité různé přesměrování. Příkladem může být přesměrováni uživatele na seznam úkolů, po úspěšném vytvoření nového úkolu. Takže změna adresy se provede v případě, kdy uživatel není přihlášen a bude přesměrován na stránku s registrací nebo přihlášením.

```
export default new VueRouter({
  mode: 'history',
  routes : [
    { path:'/google/callback/', name:'google-api',
      component:GoogleApi,
      meta:{ requireAuth:false, header:'Google' } },
    { path:'/login/', name:'login', component:Login,
      meta:{ requireAuth:false, header:'Login' } },
    { path:'/register/', name:'register',component:Register,
      meta:{requireAuth:false, header:'Register' } },
    .
    .
    .
```

Listing 3.3: Aplikační router.

Kapitola 4

Serverová část aplikace

Při návrhu aplikace bylo rozhodnuto využít třívrstvovou architekturu [7]. Prezentační vrstva a částečně aplikační jsou popsány v kapitole 3. Další vrstvy jsou databázová a serverová aplikační. Serverová aplikační vrstva zpracovává požadavky uživatele a zajišťuje všechny operace a výpočty. Databázová nebo datová vrstva pracuje s daty, tedy provádí operace výběru, ukládání, agregaci, integritu a audit dat. V této kapitole bude popsán návrh dvou posledních vrstev a jích následná implementace.

4.1 Architektura

Návrh serverové části aplikace se dá shrnout do několika fází. První etapou je stanovení modelů, které budou existovat v aplikaci, na základě specifikace požadavků uživatelů. Následně je pak pro každý z těchto modelů nutné nadefinovat interakční vztahy a individuální atributy. Výsledkem této fáze je ER (Entity–relationship) diagram [6], který představuje schéma znázorňující návrh databázové architekturu aplikace. V průběhu druhé fází se navrhuje business logika aplikace, což znamená popis implementace jednotlivých procesů, které jsou požadovány prezentační vrstvou [19]. Předposlední fáze je rozhodování o výběru implementačních prostředků a následná realizace výsledku z prvních dvou fází. Poslední etapa je testování výsledné aplikace.

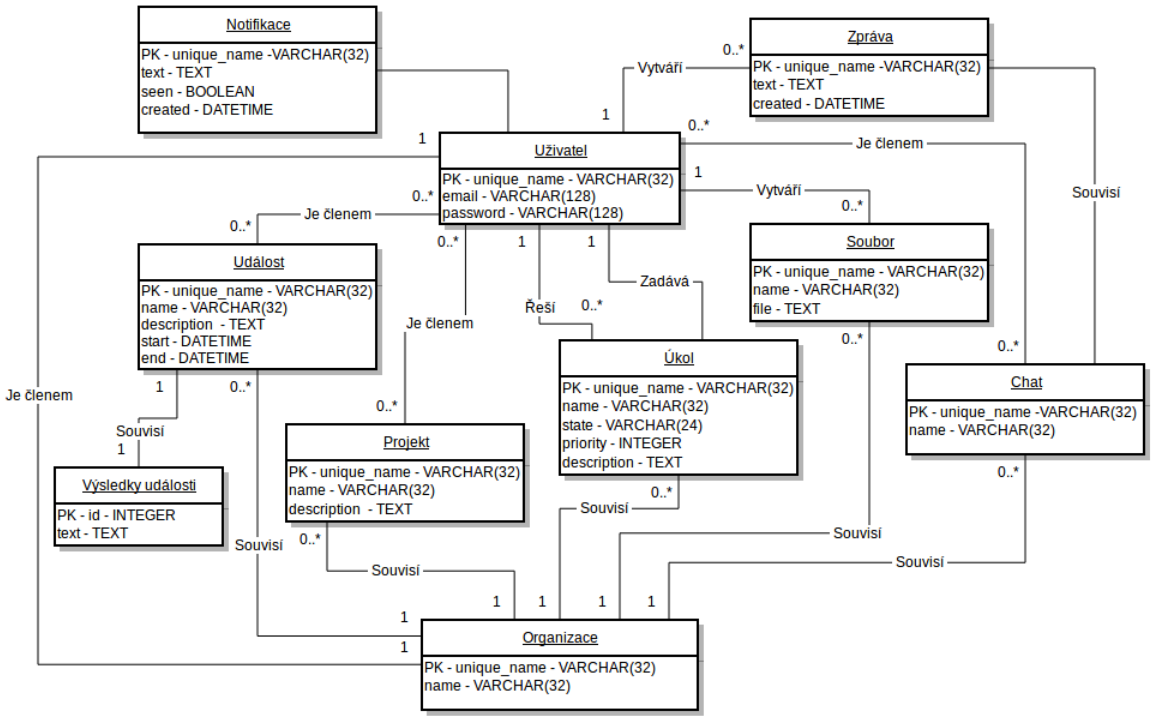
4.1.1 Databázové modely

Dříve než budou navrženy databázové modely je nutné nadefinovat abstraktní modely, které budou v systému existovat.

- Firma, společnost, organizace nebo podnik. Ve skutečnosti firma může mít adresu, a další parametry, ale pro danou aplikaci bude postačující jen její název. Tento model měl by zajišťovat kompletní datovou uzavřenost a privátnost v rámci jedné společnosti.
- Projekt. Množina atributů projektu se skládá z názvu, popisu a skupiny uživatelů, jeho členů. Navíc s projektem musí souviset množina úkolů, která je potřeba splnit, aby projekt bylo možné označit za dokončený.
- Úkol. Jeho model obsahuje název, popis úkolu, zodpovědného člena týmu, prioritu a aktuální stav.

- Událost. Model, který představuje schůzku, nebo jakoukoliv další událost, která nemá návaznost na projekt nebo úkol. Základními atributy jsou název, popis a členové týmu, kteří by se měli této události zúčastnit.
- Uživatel. Pro danou aplikaci bude postačující jeho jméno, příjmení a email. Vzhledem k tomu, že jedním z cílů aplikace je dodržovat datovou uzavřenost, uživatel by měl vždy mít minimálně a maximálně jednu návaznost na nějakou firmu.
- Soubor. Jeho atributy můžou byt název a cesta na disku.
- Chat. Představuje privátní chat pro komunikaci členů týmu. Počet členů v chatu není omezen.
- Zpráva. Jednotka komunikace v chatu. Vztahuje se k uživateli, který jí vytvořil a k chatu, do kterého ji poslal.
- Oznámení. Představuje zprávu o změně stavu objektu. Vztahuje se k uživatelům, kterých se změněný objekt týká.
- Změna. Představuje změnu události nebo úkolu. Vztahuje se k změněnému objektu, k uživateli, který změnu provedl a k času změny. Z množiny změn je postavená časová osa objektu.

Pomoci těchto modelů jsem si nadefinoval všechny vztahy v aplikaci. Výsledný ER diagram je zobrazen na obrázku dole.



Obrázek 4.1: ER diagram aplikace.

4.1.2 REST a komunikace v reálném čase

Po domluvě s vedoucím bylo rozhodnuto, že aplikace bude mít architekturu tlustého klienta. Aplikační vrstva v něm je rozdělena na dvě části. První část běží na klientovi a na server posílá jenom informace o stavecj objektů. Druhá je představena aplikací, která zpracovává požadavky o těchto stavech, viz obrázek 2.1. Příkladem může být Google Mail. Výhoda takových aplikací je v tom, že server nepotřebuje drahý a výkonný hardware, ale potřebuje ho klient. Obvykle komunikace se serverem je v stylu RESTfull. Rozhraní REST je použitelné pro jednotný a snadný přístup ke zdrojům (resources). Zdrojem v aplikaci jsou modely. Všechny zdroje mají vlastní identifikátor URI a REST definuje čtyři základní metody pro přístup k nim.

- Základní metodou pro přístup ke zdrojům je získání zdroje – metoda GET.
- Pro vytvoření dat slouží metoda POST.
- Zdroj lze smazat pomocí volání URI HTTP metodou DELETE.
- Operace změny je podobná operaci vytvoření s tím rozdílem, že voláme konkrétní URI konkrétního zdroje, který chceme změnit, a v těle předáme novou hodnotu. U úprav zdroje je jeho URI již známá. Používá se metoda PATCH.

Pro komunikaci v reálném čase byl navržen systém jednoduchých chatů. Hlavním cílem bylo poskytnout uživatelovi možnost vytvářet jak privátní, tak i skupinové chaty. Přes chaty je ale schopen posílat jenom textové zprávy. Další funkce je audio–video konference.

4.2 Implementace

V průběhu teto kapitole je popsána implementace databázového návrhu, REST a komunikace v reálném čase. Mezí tím jsou definovány další součásti aplikace, jako autorizace a export do Google kalendáře.

4.2.1 Použité technologie

Při vývoji komplexních projektů jsem měl možnost využít existujících knihoven. Dále jsou popsaný jenom ty nejdůležitější z nich.

Django

Django[1] je webový framework napsaný v jazyce Python[2]. Je zaměřen na rychlé realizace koncepce a snaží se mít v průběhu vývoje co nejvíc kroků automatizovaných. Obsahuje vlastní ORM (Object relation mapper), který umožňuje definovat databázové tabulky pomocí Python modelů. Navíc obsahuje administrační rozhraní generováne automaticky podle modelů, které vývojář potřebuje editovat. Šablonovací systém slouží k oddělení zobrazení dat od jejich získávání. Django podporuje vícejazyčné aplikace a cachování stránek, části stránek, nebo výsledků API dotazů. Celý framework je open source [16].

[1] Django – https://www.djangoproject.com/
[2] Python – https://www.python.org/

TastyPie

Tastypie[3] je aplikace pro Django, která umožňuje vytvářet REST API [17]. Obsahuje široké možností pro konfigurace REST komunikace, budování zdrojů na základě modelů, automatizace autorizace požadavků, specifikace oprávnění uživatelů a generaci URL podle zdrojů.

Django Channels

Channels[4] je aplikace, která umožňuje Django zvládat víc než jenom obyčejné HTTP požadavky. Přidává možnosti pro komunikace přes WebSocket a HTTP–2. K tomu navíc přibývá možnost pokračovat ve zpracovaní dat nebo výpočtu po odeslání odpovědi uživateli.

SQLite

SQLite[5] je relační databázový systém napsané v jazyce C. Je vyvíjen D. Richardem Hippem a šířen pod licencí public domain. Charakteristickými prvky systému SQLite jsou:

- Absence databázového systému ve formě abstrahovaného prostředí.
- Absence serveru.
- Absence konfigurace.
- Databáze v jednom souboru, nezávislém na platformě.

Google API

Google API[6] je množina aplikačních programovatelných rozhraní vývijených společností Google, které umožňují komunikaci se službami Google a integraci do dalších služeb. Příklady takových jsou Google vyhledávání, Gmail, Google Překladač a Google Mapy. Aplikace třetích stran mohou využívat tuto funkcionalitu a rozšiřovat své možnosti. API poskytuje použíti analytiky, strojového učení nebo časteční přistup k uživatelským datům.

4.2.2 Tvorba modelů

Všechny modely popsané v teoretickém návrhu 4.1.1 a byly implementovány pomocí Django. ORM (Objektově–Relační Mapování) systém v Django je mocný nastroj představení abstraktních a teoretických modelů v databázi. Pro realizace aplikace byla vybrána databáze SQLite, protože se celá databáze ve finále nachází v jednom souboru. Po implementaci teoretických modelů k nim byli přidány další pomocné, a ve výsledku projekt obsahuje 14 modelů.

- Organizace.
- Uživatel.
- Projekt.

[3] Tastypie – http://tastypieapi.org/
[4] Channels – https://github.com/django/channels
[5] SQLite – https://www.sqlite.org/
[6] Google API – https://developers.google.com/apis-explorer/

- Ukol.
- Událost.
- Soubor.
- Změna.
- Notifikace o změně.
- Notifikace o zpravě.
- Model, ve kterém jsou uloženy informace o exportu události do Google Kalendáře.
- Model, ve kterém jsou uloženy autorizační informace uživateli do Google účtu.
- Chat.
- Zpráva.
- Model, ve kterém jsou uložené informace o tom, kdy naposledy uživatel navštívil chat.

Vhledem k tomu, že podle návrhu 4.1 má každý model primární klič *unique_name* a odkaz na organizaci, jsem vytvořil další abstraktní model. Navíc k němu byli přidány dva atributy, *creator* je uživatel, který objekt vytvořil, *created* je čas vytvoření. Primární klíč představuje náhodnou posloupnost znaků s celkovou délkou 32. Tento model byl zděděn do ostatních, kde bylo potřeba tyto atributy používat, a je představen dole:

```
class MainAbstractModel(models.Model):
    unique_name = models.CharField(max_length=32)
    creator = models.ForeignKey('FitbcUser', null=True)
    created = models.DateTimeField(auto_now_add=True)
    organization = models.ForeignKey('Organization', null=True)

    def save(self, *args, **kwargs):
        if not self.pk:
            self.unique_name = get_random_string(length=32)
```

Listing 4.1: Část abstraktního modelu se společnými atributy.

Dále je uveden jednoduchý příklad implementace modelu Soubor pomocí Django. Model obsahuje jméno, které bude v databázi spojeno s jménem aplikace, ve které je tento model definován, a z toho Django vytvoří jméno tabulky. Tento princip umožňuje vytvářet modely se stejnými jmény v různých aplikacích, ale tato strategie vývoje není doporučována. Dále jsou popsané všechny atributy v modelu, které se vyskytují v tabulce jako sloupce. V třídě *Meta* jsou uložené užitečné informace pro styl práce s modelem. *Verbose_name* je zobrazení názvu modelu v administračním rozhraní. Dalšími atributy můžou byt pořadí při filtraci, název tabulky v databázi atd. Posledním je definovaná metoda pro normalizaci textového zobrazení instance modelu. Navíc je k tomu možné přidávat další pomocné metody, ale vždy by se mělo pamatovat, že ukládání modelu může probíhat i bez přístupu uživatele. Na to jsem narazil při realizaci části aplikace zodpovědnou za tvorbu objektů změn. Problém spočíval v tom, že ne vždy při využití modelové metody pro ukládaní do databáze objektů aplikace měla přístup k uživatelovi, který změnu provedl. Po provedení malého výzkumu bylo rozhodnuto přenést tvorbu změn do části REST.

```
class File(models.Model):
    name = models.CharField(max_length=32)
    file = models.FileField(upload_to='media/')

    class Meta:
        verbose_name = "File"
        verbose_name_plural = "Files"

    def __str__(self):
        return self.name
```

Listing 4.2: Příklad implementaci modelu pomoci Django.

Během migrace modelů do databáze Django vytváří pomocné systémové modely, které jsou nezbytné pro správnou funkčnost frameworku. V této sekci jsou uvedeny techniky a modely použité při realizaci historie změn.

- Model, ve kterém je uložená historie změn všech modelů v projektu [13]. Je využit v administrační části aplikace.
- Model, ve kterém jsou uloženy všechny typy modelů existujících v systému. [11]

Poslední model je důležitý z toho důvodu, že umožňuje provádět zajímavé operace. Pokud je třeba, aby některý náš model odkazoval na několik dalších modelů v aplikaci, tak bychom během tvorby databázového návrhu museli vytvořit tolik cizích klíčů, kolik je modelů, na které potřebujeme odkázat. Příkladem je model změn. Ten je možné vytvořit pro úkol nebo událost, a tak by bylo nutné vytvořit dva cizí klíče. Ale Django umožňuje realizaci jiným způsobem. V modelu změny je potřeba definovat jeden cizí klič na tento systémový model, popsaný výše, a jedno pole pro ukládání hodnoty primárního klíče modelu, na který je potřeba odkázat. Cizí klič bude v tom případě obsahovat textový název modelu, na který je třeba odkázat, a spolu s hodnotou primárního klíče tvoří generický cizí klič [12]. Generický je kvůli tomu, že v různém čase může odkazovat na různé tabulky. Musíme ale brát v potaz, že typ hodnoty primárního klíče musí být stejný ve všech modelech, na které budeme odkazovat. Na druhou stranu, kvůli této vlastnosti bude omezena možnost filtrace a vývojář musí vždy vědět jaký model potřebuje vyhledat.

4.2.3 REST komunikace

Při realizaci REST byla využita knihovna TastyPie. TastyPie umí vytvořit Resource(dále zdroj), který bude rovnou navázán na Django model.

```
class NotificationResource(ModelResource):
    class Meta:
        resource_name = 'notification'
        queryset = Notification.objects.all()
        authorization = Authorization()
        authentication = SessionAuthentication()
        fields = ['text']
        detail_uri_name = 'unique_name'
    def get_object_list(self, request):
```

```
return request.user.notification_set.filter(seen=False)
```

Listing 4.3: Příklad TastyPie zdroje

Zdroj je navázány na adresu, která se skládá ze dvou částí. První je adresa API, která je uvedena v konfiguračním souboru, a druhá je jméno zdroje, které je uvedeno v atributu *resource_name*. Ve zdroji je navíc popsána základní filtrace a zároveň i Django model, na který bude zdroj napojen. Navíc může být vybrán typ autorizace nebo autentizace uživatele 4.2.4. Vrácení výsledku může být ve formátech JSON nebo XML.

Princip funkčnosti TastyPie spočívá v tom, že požadavek od klienta vždy prochází dvěma etapama. V první etapě server vytahuje z požadavku informace identifikující instance modelu definovaného ve zdroji, nebo seznam instancí modelu a s ním se provádí potřebné operace. Nejčastější operací je filtrace. V průběhu druhé etapy se každá instance ze seznamu připravuje pro odeslání. Vybírá se z něj pole, které bylo definováno ve zdroji a zpracuje se do výsledného formátu JSON nebo XML. Během tohoto kroku je možné přidávat vlastní textová data, která nebyla definována ve zdroji.

V aplikaci bylo potřeba mít univerzální model, který by obsahoval historii změn. Při řešení byl využit systémový Django model pro tvorbu speciálního generického cizího klíče a textu s popisem změny. Proto se aktualizace, tvorba a mazání objektů provádějí jenom ve zdrojích,kde byla vytvořena třída, která přepisovala tyto tři operace a vytvářela instance modelu změn. Tato třída byla následně zděděna do všech zdrojů, ve kterých bylo dohodnuto ukládání historie. Zobrazení historie v klientovi bylo realizováno na časové ose. V množině systémových Django modelů existuje model pro ukládání historie, ale v něm se neukládají původní hodnoty, které byly v objektu změněny.

Notifikace o změně nebo vytvoření objektu je třeba realizovat jen v některých zdrojích a je vytvářena na základě speciálních událostí. Notifikace představuje informaci o změně nebo tvorbě objektu a rozesílá se všem uživatelům, kteří jsou s tímto objektem propojeni. V aplikaci existují notifikace popisující změnu projektu, úkolu nebo události.

4.2.4 Autorizace a autentizace uživatelů

V Django existuje modul, který obsahuje modely pro autorizaci, autentizaci a řízení oprávnění uživatelů [10]. V nich je definován model uživatele, změna nebo rozšíření, které musí být provedeno s opatrností. Pro správnou změnu existují dvě možnosti. První je vytvoření modelu, který bude obsahovat všechna pole, o které je potřeba standardní model rozšířit a navíc odkaz (cizí klič) na standardní model uživatele. V takovém případě je potřeba počítat s tím, že při každém volání do databáze bude přibývat ve filtru jeden JOIN, kvůli propojení dvou modelů. Tato varianta je časově náročná při velkém množství uživatelů a stálé komunikace se serverem. Druhá varianta je změna standardního modelu na vlastní. Standardní model Django je zděděn z abstraktního modelu uživatele, který obsahuje všechny běžné atributy, jako například jméno nebo příjmení. Navíc obsahuje definice metod jako registrace, přihlášení atd. Pro nahrazení byl vytvořen nový model uživatele, následně zděděn z abstraktního modelu a doplněn o potřebné atributy a metody, což v projektu byli primární klíč a odkaz na model organizace.

V aplikace existuje rozšířena registrace. Podle teoretického návrhu aplikace má za úkol zjednodušit skupinovou registraci uživatelů. Proto při registraci existuje možnost zadat emailové adresy několika uživatelů najednou. Každému pak bude vytvořen účet, který bude obsahovat uživatelské jméno a heslo stejné s hodnotou emailové adresy. Změnu hesla je možné provádět přes administrační rozhraní.

Existuje několik možností kontroly autentizace uživatelů pomoci TastyPie. Základní možnosti jsou:

- Authentication – žádná kontrola. Je vhodná pro otevřené zdroje.
- BasicAuthentication – kontrola údajů uživatele v každém požadavku.
- ApiKeyAuthentication – kontrola API klíče v každém požadavku.
- SessionAuthentication – využity standardní autentizace Django frameworku.

Pro autentizaci všech požadavků na API byl vybrán standardní systém Django autentizace. Po autentizaci uživatele k němu do „HTTP coockie“ se přidává unikátní klič pro jeho identifikaci. Pro autentizaci byl vytvořen Django pohled, který přijímal požadavky od uživatele a prováděl autentizace.

Pro autorizaci existuje několik možnosti:

- Authorization – žádná kontrola povolení ke čtení nebo zápisu. Vzhledem k tomu, že v projektu neexistuje zakázání změny nebo mazaní objektů byla vybrána tato možnost.
- ReadOnlyAuthorization – povoleno jenom čtení objektů.
- DjangoAuthorization – využita povolení Django.

4.2.5 Export události do Google kalendaře

Google využivá autorizace typu OAuth2.0. Vývojář musí udělat několik věcí, aby dostal přistup k API Google. Prvním krokem je vytvořit Google účet, pak následuje vytvoření API klíče, který jeho aplikaci unikátně identifikuje v databázi Google aplikací. Poslední krok bude napojit povolení čtení a zápis do Google Kalendáře na vytvořený klíč. Dále existuje možnost využít již existující knihovnu pro Python, která usnadňuje práci s Google API. V tomto projektu je realizována možnost exportu události do Google Kalendáře. Na začátku musí uživatel povolit externí aplikaci, provést přidání nových událostí do jeho kalendáře. V případě, že uživatel předtím nikdy nedával povolení nebo doba platnosti povolení vypršela, musí znovu provést autorizaci přes Google. Uživatelské povolení bude uloženo v databázi. Druhým a hlavním krokem je zaslání popisu události do Google API s odkazem na kalendář uživatele a popisem, reprezentovaných přes JSON objekt.

4.2.6 Komunikace v reálném čase

Komunikace v reálném čase v projektu je rozdělená na dvě části. První je komunikace přes textové zprávy. Druhou část tvoří video hovory.

Textová komunikace

Textová komunikace je realizována pomocí WebSocket. Princip odesílání zpráv je následující. Společně s každou zprávou se posílá i unikátní indentifikator chatu. Na serveru je zpráva propojená s aktuálním uživatelem přes jeho session a ukládá se do databáze. Dál se zpráva posílá všem uživatelům, kteří jsou taky momentálně napojení na tento chat. Pokud existuje někdo, kdo se nenachází na stránce chatu, ale na nějaké jiné stránce, tak se pro tyto uživatele vytváří notifikace o nové zpravě. Tato notifikace obsahuje email uživatele, který

zprávu poslal, text, unikátní indentifikator místnosti, kam zpráva patří a datum vytvoření. Na serveru o připojení přes WebSocket byla využita knihovna Django Channels. Základním prvkem této knihovny je channel nebo kanál, což je fronta FIFO, do které se posílají všechny zprávy. Posílat je může kdokoliv, ale výbirat této zprávy může maximálně jeden proces. Tak byly definovaný tři kanály podle třech stavů WebSoketa, propojení, přeposlání zpráv a odpojení od serveru. Jak je uvedeno výše, při přijetí zpráv je možné využít session, která umožňuje zjistit od jakého uživatele zpráva přišla. Tato informace pomůže zjistit, jestli ten uživatel má přistup k chatu, do kterého poslal zprávu.

Video hovory

WebRTC je popsán v RFC 7478 [8]. Tato technologie umožňuje vytvářet videohovory mezi uživateli napřímo a přeposilat soubory a text. Pro to, aby uživatelé zjistili své adresy a jiné potřebné informace a vyměnili si je mezi sebou, potřebují signální server. Uživatelé mohou posílat různé signály na tento server, například ohledně toho, že chtějí začít WebRTC komunikaci nebo ji ukončit atd. Postup pří zahájení video hovoru je následující. První uživatel vytváří objekt, ve kterém ukládá popis své sítě, a další konfigurační údaje a posílá ho druhému uživateli přes signální server. Ten druhý si tyto informační údaje ukládá a posílá své konfigurace prvnímu. Popis je velmi zjednodušen a podrobněji ho můžeme najít na této stránce [22]. Dále aby uživatelé mohli zjistit své veřejné adresy, je použit STUN server. A pokud připojení spadne, nová cesta bude procházet přes TURN server. Všechny tyto věci poskytuje knihovna SimpleWebRTC. Princip komunikace spočívá v tom, že pro skupinu uživatelů se vytvoří místnosti (rooms), do kterých se mohou připojovat a provádět komunikaci.

Kapitola 5

Testování

Testování je nezbytnou součástí vývoje jakéhokoliv projektu. V tomto projektu byly použity dva typy testování. Funkcionální testování, které je zodpovědné za ověření funkčnosti aplikace, a testování uživatelského rozhraní, které testuje jak rychle a efektivně uživatel zvládá úkoly, a jak je pro něj rozhraní uživatelsky přívětivé.

5.1 Testování funkčnosti

Aplikaci je možné rozdělit na tři logické celky. Všechno, co je spojeno s autorizací, autentizací a změnou vlastních údajů uživatele, je možné zařadit do první části. Hlavním důvodem je to, že tato část nebyla realizována pomocí REST API. Testování odhalilo několik chyb souvisících s registrací skupiny uživatelů. Samotné testy kontrolovaly funkčnost registrace, přihlášení, odhlášení a změny údajů uživatele. Druhou část tvoří REST API. Z důvodu toho, že v aplikaci existoval systém sledování stavu objektu, který vytvářel notifikace a objekty změn, skupina testů pro tuto část byla nejrozsáhlejší. Testovala se tvorba, aktualizace a mazání objektů přes REST API, a následná kontrola objektů změn. Navíc se kontroluje reakce aplikace na nepřihlášené uživatele. Poslední část aplikace tvoří komunikace v reálném čase. Vytvořena skupina testů pro kontrolu funkčnosti této části je zodpovědná za kontrolu funkcí odeslání a přijetí textových zpráv. Každá z využitých externích knihoven navíc obsahuje své vlastní testy, které kontrolují funkčnost samotných knihoven.

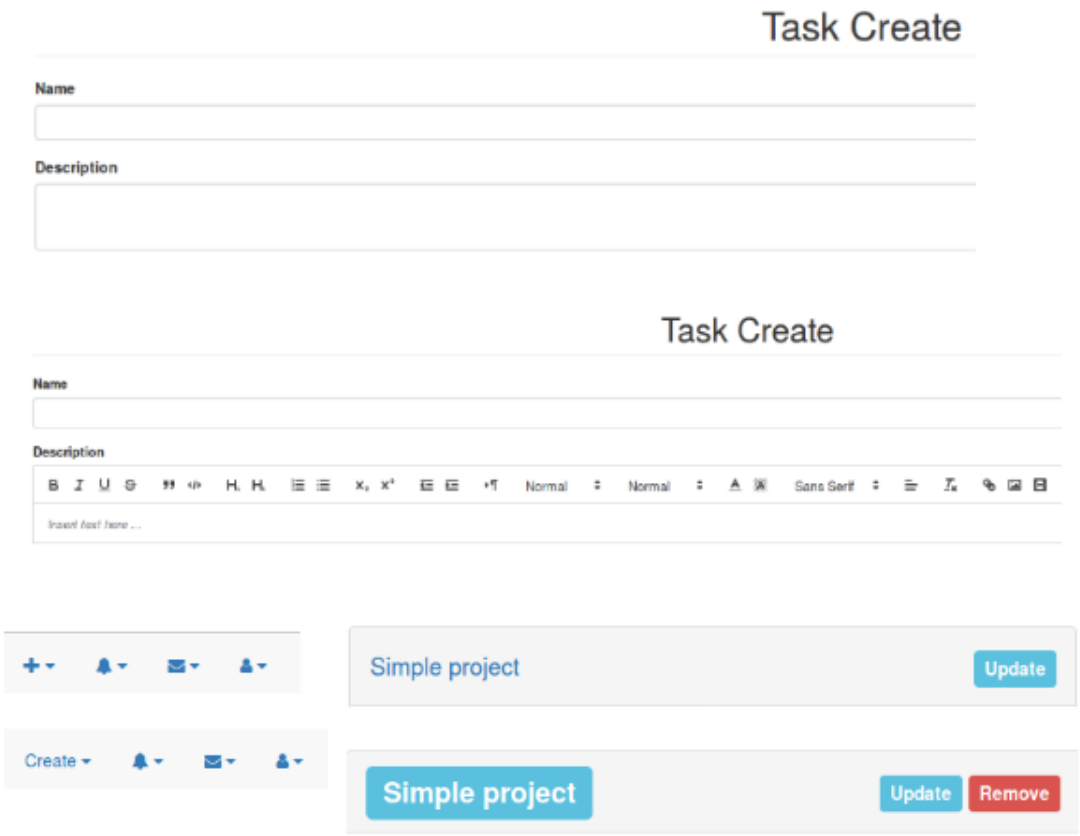
5.2 Testování uživatelského rozhraní

Pro tento typ testování bylo pozváno pět osob, jejichž úkolem bylo vyřešit a provést několik jednoduchých úloh s využitím mé aplikace. Níže je uveden seznam úkolů.

- Registrace.
- Přihlášení.
- Změna údajů.
- Tvorba projektů, úkolů a událostí.
- Změna těchto objektů a mazání.
- Tvorba události a následný export do Google Kalendáře.

- Odeslání zpráv ostatním uživatelům.
- Videohovor.
- Přidávání a prohlížení souborů.

Během testování nevznikly u testujících téměř žádné problémy s prováděním úkolů. Diky tomu, že uživatelské rozhraní je jednoduché, testující splnili úlohy za velmi krátkou dobu. Ve výsledku testování uživatelského rozhraní bylo provedeno několik zlepšení samotné aplikace. Příklady jsou zobrazeny na obrazcích 5.1.



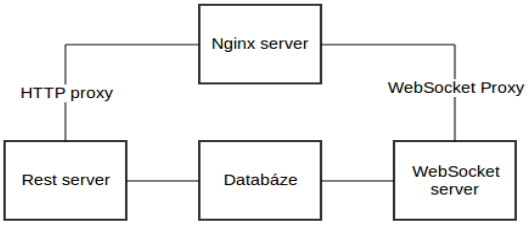
Obrázek 5.1: Příklady změn uživatelského rozhraní.

Kapitola 6

Provoz aplikace

Aplikace je přístupná na adrese https://fitbc.cf/ až do konce června 2017 roku. Výsledná organizace serveru je znázorněna na obrázku 6.1. Dál jsou uvedeny jednotlivé kroky s detailním popisem.

- Pronájem serveru. Server je poskytován společností DigitalOcean[1]. Na serveru je nainstalován operační sytém Ubuntu verze 16.10 (Yakkety Yak).
- Instalace potřebných knihoven pro Nginx [15], a samotný web server.
- Rozběhnutí Git repozitáře a vývojových serverů pro testování přístupnosti.
- Pronájem domény „fitbc.cf“. Doména byla zaregistrována přes stránku Freenom[2]. Hlavní výhodou bylo to, že nebylo potřeba za ní platit, ale využití této domény je časově omezeno na tři měsíce.
- Jelikož WebRTC komunikace v Google Chrome prohlížeči potřebuje zabezpečené spojený (HTTPS), bylo nutné vygenerovat SSL/TLS certifikát. Realizace této operaci proběhla pomocí systému „Let's Encrypt[3]“.
- Jak je vidět na obrázku, datový server je spouštěn zvlášť, oproti WebSocket serveru. Aby toto bylo možné zprovoznit na Nginx, byli nastaveny proxy [14] pro přeposlání všech požadavků na jednotlivé servery.



Obrázek 6.1: Schéma produkčního nastavení serveru.

[1] DigitalOcean – https://www.digitalocean.com/
[2] Freenom – http://www.freenom.com
[3] Let's Encrypt – https://letsencrypt.org/

Kapitola 7

Závěr

Cílem teto bakalářské práce bylo vytvořit aplikaci pro týmovou spolupráci. Na začátku byly definovýny požadavky uživatelů a kritéria takových aplikací, a následně byla provedena jejich specifikace. Při porovnání existujících neplacených populárních řešení nebyly nalezeny optimální, které by mohli splnit požadavky pro řešení problémů týmové kolaborace. Jedním z hlavních cílů aplikace bylo dosáhnout uzavřeného prostoru pro vývoj projektů a také poskytování všech základních funkcí pro splnění požadavků řešení problému. Nasledně byly navrženy prezentační, aplikační a databázové vrstvy. Prezentační vrstva byla představena jako jednostránková aplikace. Pro něj byl navržen design uživatelského rozhraní a ten byl následně implementován. Vzhledem k tomu, že jednostránkové aplikace představují částečně i aplikační vrstvu, serverová část aplikační vrstvy je pak představena REST aplikací, která jenom přijímá požadavky od jednostránkové aplikace a synchronizuje stav s databází. Pro databázi, která představuje datovou vrstvu, byli navržené modely a následně pomocí ORM (Objektově relační zobrazení) systému Django byli přeneseny na korespondující databázové tabulky. Na konci byli všechny tři vrstvy otestovány.

Během vývoje jsem musel řešit problémy jako návrh architektury aplikace, návrh databázového schématu, výběr jazyka pro realizaci, výběr frameworku jak pro databázové pozadí (backend), tak i pro tvorbu uživatelského rozhraní. Navíc byl k tomu v projektu navržen jednoduchý design, zjištěn princip použití knihoven pro komunikaci v reálném čase a integrace aplikací třetích stran jako Google Kalendář a SimpleWebRTC. Posledním, ale taky důležitým úkolem, bylo nasadit aplikaci do provozu.

Další vylepšení a rozšíření výsledné aplikaci byli navrženy uživateli v průběhu testování uživatelského rozhraní.

- Pokud existuje možnost videohovoru, tak by mohla být i možnost ukázat vzdálenou plochu.
- Notifikace v systému by se mohli zasílat periodicky do emailu.
- Přidání možnosti odesílání souboru přes instantní zprávy.

Ve výsledku jsem vytvořil komplexní projekt, který může být dobrým základem nebo příkladem pro řešení problémů týmové spolupráce.

Literatura

[1] Basecamp : Basecamp. [Online; navštíveno 18.02.2017].
URL https://basecamp.com/

[2] Fog Creek Software : Trello. [Online; navštíveno 18.02.2017].
URL https://trello.com/

[3] Butterfield, S.: Slack. [Online; navštíveno 18.02.2017].
URL https://slack.com/

[4] CodeKick: KanbanFlow. [Online; navštíveno 18.02.2017].
URL https://kanbanflow.com/

[5] Dennis, A.; Wixom, B. H.; Roth, R. M.: *SYSTEM ANALYSIS AND DESIGN*. John Wiley & Sons, Inc., páté vydání, 2012, ISBN 978–1–118–05762–9.

[6] Elmasri, R.; Navathe, S. B.: *Fundamentals of Database Systems*. Addison-Wesley Longman Publishing Co., 6 vydání, 2011, ISBN 978–0–136–08620–8, 199 - 245 s.

[7] Gemela, L.: *Možnosti návrhu webových aplikací*. Plzeň, 2013.

[8] Holmberg, C.; Hakansson, S.; Eriksson, G.: Web Real-Time Communication Use Cases and Requirements. [Online; navštíveno 18.02.2017].
URL https://tools.ietf.org/html/rfc7478

[9] Johnston, A.; Yoakum, J.; Singh, K.: *Taking on webRTC in an enterprise*. IEEE Communications Magazine, April 2013, doi:10.1109/MCOM.

[10] Kolektiv autorů: Django auth module. [Online; navštíveno 12.03.2017].
URL https://docs.djangoproject.com/en/1.10/topics/auth/

[11] Kolektiv autorů: Django ContentType model. [Online; navštíveno 12.03.2017].
URL https://docs.djangoproject.com/en/1.10/ref/contrib/contenttypes/#the-contenttype-model

[12] Kolektiv autorů: Django Generic Relations. [Online; navštíveno 12.04.2017].
URL https://docs.djangoproject.com/en/1.10/ref/contrib/contenttypes/#generic-relations

[13] Kolektiv autorů: Django LogEntry objects. [Online; navštíveno 12.03.2017].
URL https://docs.djangoproject.com/en/1.10/ref/contrib/admin/#logentry-objects

[14] Kolektiv autorů: Nginx reverse proxy. online.
URL https://www.nginx.com/resources/admin-guide/reverse-proxy/

[15] Kolektiv autorů: Nginx server. online.
URL https://www.nginx.com/

[16] Kolektiv autorů: Open source software. [Online; navštíveno 12.03.2017].
URL https://cs.wikipedia.org/wiki/Otev%C5%99en%C3%BD_software

[17] Kolektiv autorů: Rest. [Online; navštíveno 12.04.2017].
URL https://cs.wikipedia.org/wiki/Representational_State_Transfer

[18] McKay, E. N.: *UI is Communication: How to Design Intuitive, User Centered Interfaces by Focusing on Effective Communication*. Morgan Kaufmann, první vydání, 2013, ISBN 978–0-123–96980–4.

[19] Microsoft Corporation: *Application Architecture for .NET: Designing Applications and Services by Microsoft Corporation*. Microsoft Press, první vydání, 2003, ISBN 978–0-735–64391–8.

[20] Pimentel, V.; Nickerson, B. G.: *Communicating and Displaying Real-Time Data with WebSocket*. IEEE, May 2012, 45 - 53 str.

[21] Rumbaugh, J.; Booch, G.; Jacobson, I.: *The Unified Modeling Language Reference Manual*. Addison-Wesley Professional, druhé vydání, 2004, ISBN 03–2171–895–X.

[22] Sam Dutton: WebRTC in the real world: STUN, TURN and signaling. online, navštíveno 12.03.2017.
URL https://www.html5rocks.com/en/tutorials/webrtc/infrastructure/

[23] You, E.: Instance lifecycle hooks. online.
URL https://vuejs.org/v2/guide/instance.html#Instance-Lifecycle-Hooks

Přílohy

Příloha A

Obsah CD

- Technická zpráva (**documentation.pdf**)
- Zdrojové soubory (**fitbc.zip**)
- Kratký popis aplikaci (**README**)